Metody obliczeniowe w nauce i technice - sprawozdanie 5

Łukasz Jezapkowicz

19.04.2020

1 Mamy równanie: $f(x) = x^2 - 2 = 0$.

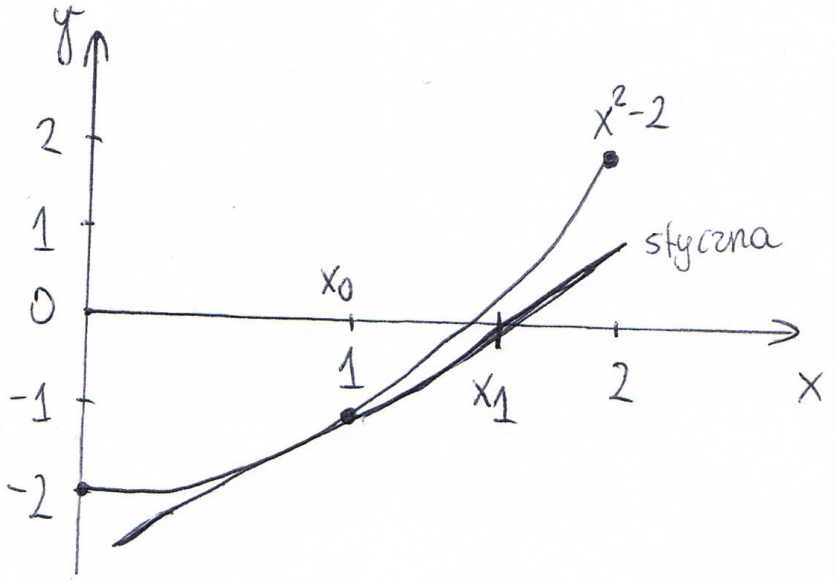
1.1 Zakładając że mamy punkt początkowy $x_0 = 1$, jaką wartość x_1 dostaniemy, jeśli używamy metody Newtona?

1.a) $f(x) = x^2 - 2 = 0$, $x_0 = 1$

$f'(x) = 2x$, $f'(x) \neq 0 \Rightarrow x \neq 0$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

$$x_1 = 1 - \frac{f(1)}{f'(1)} = 1 - \frac{-1}{2} = \frac{3}{2} = 1.5$$



1.2 Zakładając że mamy $x_0 = 1$ i $x_1 = 2$ jako punkty początkowe, jaka będzie wartość x_2, jeśli używamy metody siecznych do tego samego problemu?

1.b) $x^2 - 2 = 0$ $\quad x_0 = 1,\ x_1 = 2$

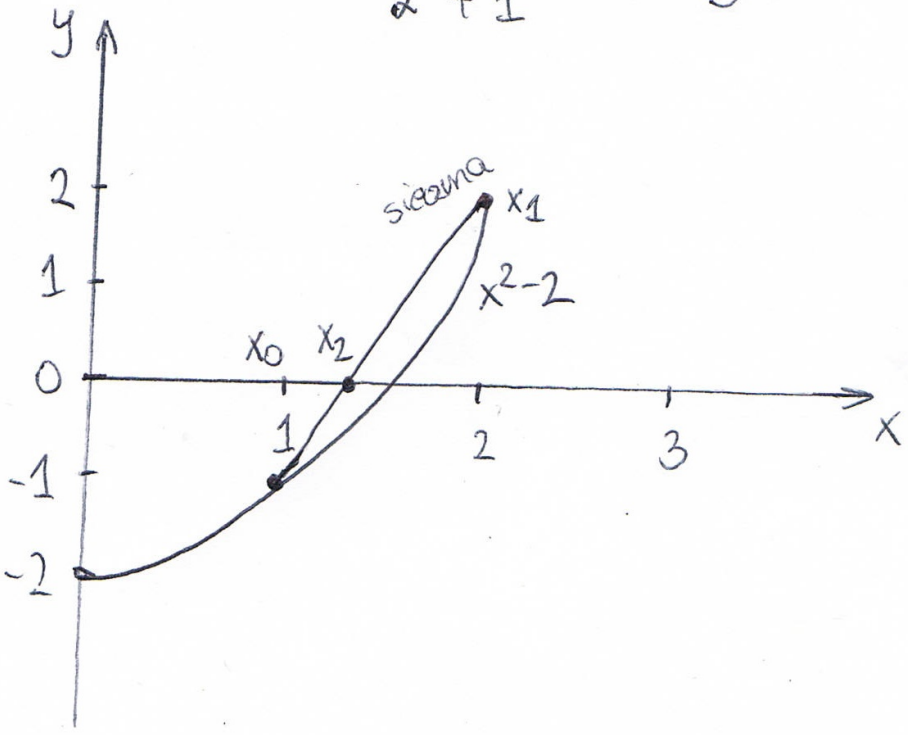
Sprawdzam warunek $f(x_0) \cdot f(x_1) < 0$

$f(1) \cdot f(2) = -1 \cdot 2 = -2 < 0$ spełniony

Wtedy:

$$x_2 = \frac{f(x_1) \cdot x_0 - f(x_0) \cdot x_1}{f(x_1) - f(x_0)}$$

$$x_2 = \frac{2 \cdot 1 - (-1) \cdot 2}{2 + 1} = \frac{4}{3} = 1.(3).$$



2 Napisz iteracje wg metody Newtona do rozwiązywania każdego z następujących równań nieliniowych:

2.1 Program w Pythonie

W zadaniu posłużyłem się napisanym wcześniej prostym algorytmem iteracyjnym wykonującym kolejne kroki metody Newtona-Raphsona. Algorytm widoczny jest poniżej:

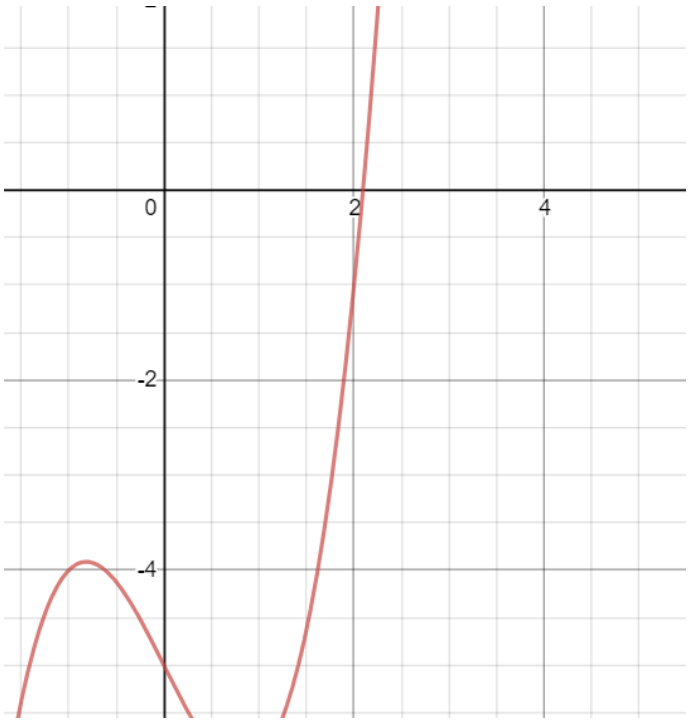
```
from sympy import *
def newton_raphson(f,der,left,right,steps):
    if left >= right or f(left)*f(right)>=0:
        print("Can't calculate root for such values!")
        return

    current = left
    for i in range(steps):
        current = current - f(current)/der(current)
        print('Value of x after ' + str(i+1) + ' iterations: x = ' + str(current))
```

Warunki jakie musi spełniać funkcja na przedziale $[a, b]$, żeby móc zastosować dla niej metode Newtona-Raphsona:

1. W przedziale $[a, b]$ znajduje się dokładnie jeden pierwiastek.
2. Funkcja ma różne znaki na krańcach przedziału, tj $f(a) * f(b) < 0$
3. Pierwsza i druga pochodna funkcji mają stały znak w tym przedziale.

2.2 $x^3 - 2x - 5 = 0$

Należy sprawdzić, dla jakiego przedziału funkcja $f(x) = x^3 - 2x - 5$ spełnia warunki dla metody Newtona-Raphsona. Poniżej wykres funkcji f przy użyciu programu Desmos:



Nie trudno zobaczyć, że miejsce zerowe jest między 2 i 3. Przyjmuję więc większy przedział $[1, 10]$ w celu lepszego spojrzenia na wyniki algorytmu. W owym przedziale funkcja ma tylko jeden pierwiastek, wartości na krańcach przedziału mają różne znaki ($f(1) = -6, f(10) = 975$) oraz pierwsza i druga pochodna mają stały znak w tym przedziale (dodatni, dodatni). Można zatem zastosować metode Newtona-Raphsona.

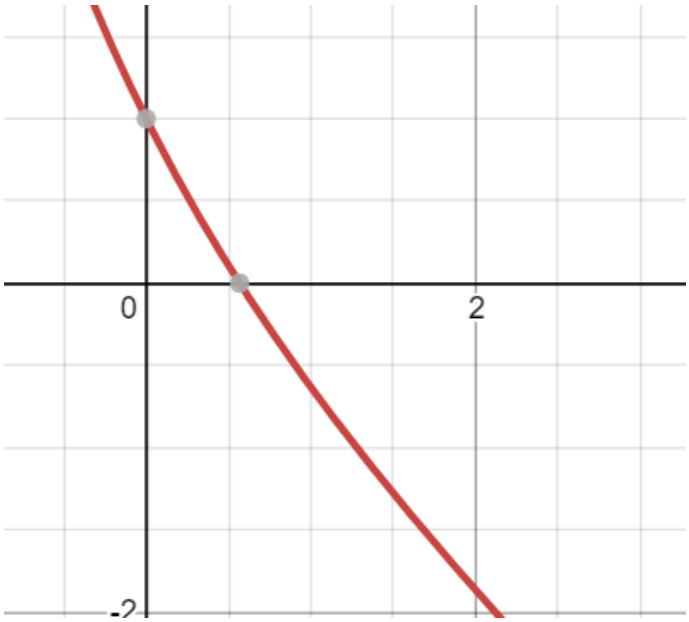
```
def f1(x):
    return x**3-2*x-5

def f1_der(x):
    return 3*x**2-2

newton_raphson(f1,f1_der,1,3,10)
```

```
Value of x after 1 iterations: x = 7.0
Value of x after 2 iterations: x = 4.76551724137931
Value of x after 3 iterations: x = 3.3487027594802825
Value of x after 4 iterations: x = 2.53159964100251
Value of x after 5 iterations: x = 2.1739158849392317
Value of x after 6 iterations: x = 2.097883686441764
Value of x after 7 iterations: x = 2.0945577158500575
Value of x after 8 iterations: x = 2.0945514815642077
Value of x after 9 iterations: x = 2.0945514815423265
Value of x after 10 iterations: x = 2.0945514815423265
```

Wolphram: $x = 2.09455148154233$

2.3 $e^{-x} = x$

Należy sprawdzić, dla jakiego przedziału funkcja $f(x) = e^{-x} - x$ spełnia warunki dla metody Newtona-Raphsona. Poniżej wykres funkcji f przy użyciu programu Desmos:



Nie trudno zobaczyć, że miejsce zerowe jest między 0 i 1. Przyjmuję więc większy przedział $[0, 2]$ w celu lepszego spojrzenia na wyniki algorytmu. W owym przedziale funkcja ma tylko jeden pier-

wiastek, wartości na krańcach przedziału mają różne znaki ($f(0) = 1, f(10) \approx -1.86$) oraz pierwsza i druga pochodna mają stały znak w tym przedziale (ujemny,dodatni). Można zatem zastosować metode Newtona-Raphsona.

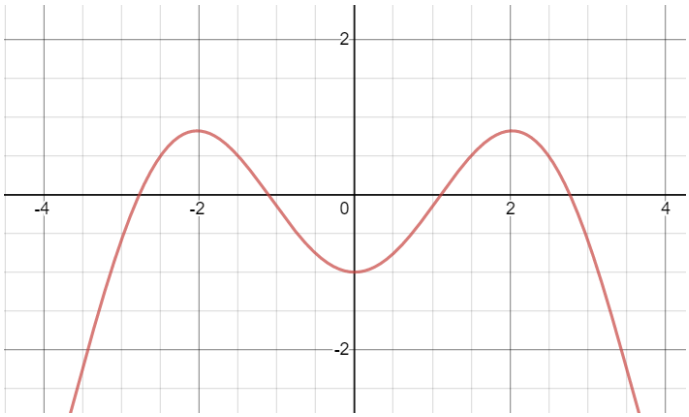
```
from math import exp
def f2(x):
    return exp(-x)-x

def f2_der(x):
    return -exp(-x)-1

newton_raphson(f2,f2_der,0,2,10)
```

```
Value of x after 1 iterations: x = 0.5
Value of x after 2 iterations: x = 0.5663110031972182
Value of x after 3 iterations: x = 0.5671431650348622
Value of x after 4 iterations: x = 0.5671432904097811
Value of x after 5 iterations: x = 0.567143290409784
Value of x after 6 iterations: x = 0.5671432904097838
Value of x after 7 iterations: x = 0.5671432904097838
Value of x after 8 iterations: x = 0.5671432904097838
Value of x after 9 iterations: x = 0.5671432904097838
Value of x after 10 iterations: x = 0.5671432904097838
```

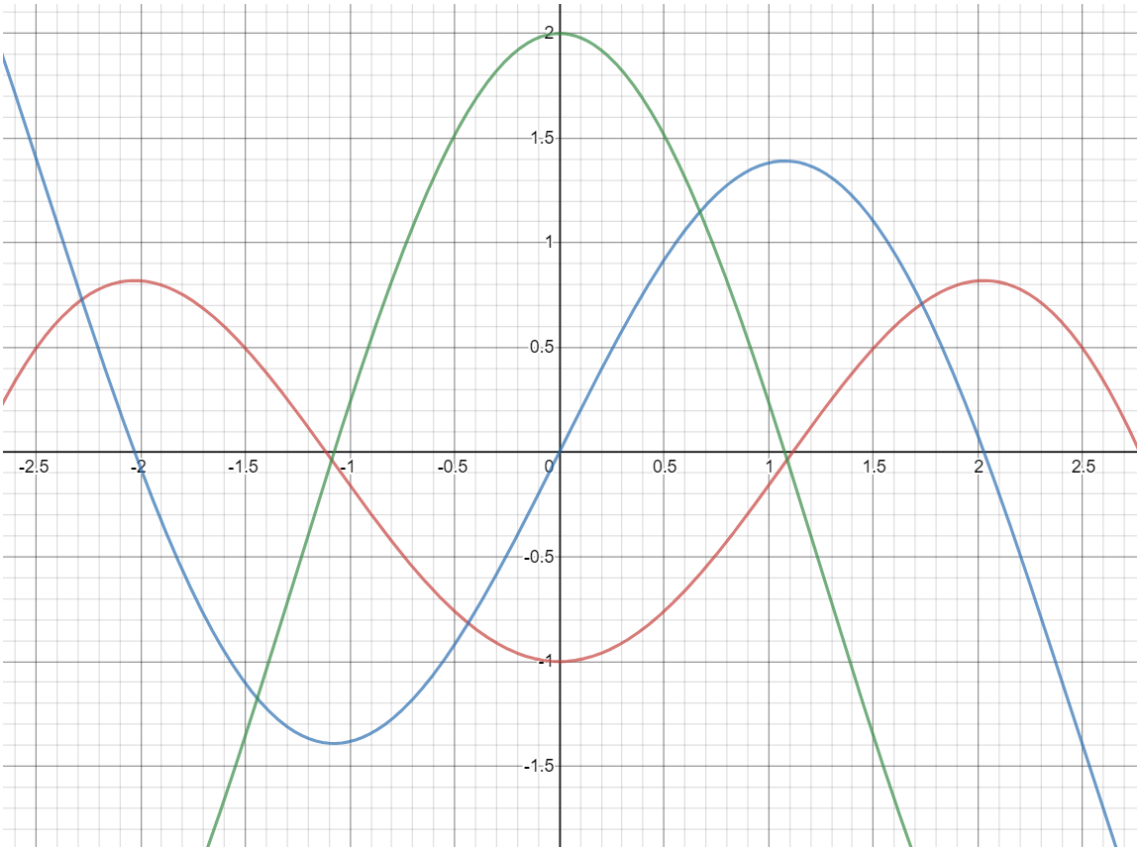
Wolphram: $x = 0.56714329040978$

2.4 $xsin(x) = 1$

Należy sprawdzić, dla jakiego przedziału funkcja $xsin(x) - 1$ spełnia warunki dla metody Newtona-Raphsona. Poniżej wykres funkcji f przy użyciu programu Desmos:



Owa funkcja jest okresowa i ma nieskończenie wiele pierwiastków. Należy więc wybrać pewien przedział, w którym istnieje tylko jeden pierwiastek. Występuje jednak problem, ponieważ nie da się przyjąć takiego przedziału, dla którego zarówno pierwsza jak i druga pochodna są stałego znaku oraz spełnione są wszystkie pozostałe warunki metody Newtona-Raphsona. Problem widać na wykresie poniżej:



Czerwony - $f(x)$, Niebieski - $f'(x)$, Zielony - $f''(x)$

Jeżeli mimo tego, zastosujemy metodę Newtona-Raphsona na przedziale np. $[0.5, 2]$ otrzymamy następujące wyniki:

```
def f3(x):
    return x*sin(x)-1

def f3_der(x):
    return sin(x)+x*cos(x)

newton_raphson(f3,f3_der,0.5,2,10)
```

```
Value of x after 1 iterations: x = 1.32800403402652
Value of x after 2 iterations: x = 1.10392072872192
Value of x after 3 iterations: x = 1.11415350604505
Value of x after 4 iterations: x = 1.11415714087137
Value of x after 5 iterations: x = 1.11415714087193
Value of x after 6 iterations: x = 1.11415714087193
Value of x after 7 iterations: x = 1.11415714087193
Value of x after 8 iterations: x = 1.11415714087193
Value of x after 9 iterations: x = 1.11415714087193
Value of x after 10 iterations: x = 1.11415714087193
```

Wolphram: $x = 1.11415714087193$

2.5 Wniosek

Metoda Newtona-Raphsona potrafi w pewnych przypadkach dać poprawny wynik nawet jeżeli nie są spełnione wszystkie potrzebne warunki. Drugim wnioskiem jest fakt, że metoda ta przybliża stosunkowo dobrze nawet dla małej ilości iteracji.

3 Zapisz iteracje Newtona do rozwiązywania następującego układu równań nieliniowych:

$x_1^2 + x_2^2 = 1,\ x_1^2 - x_2 = 0$

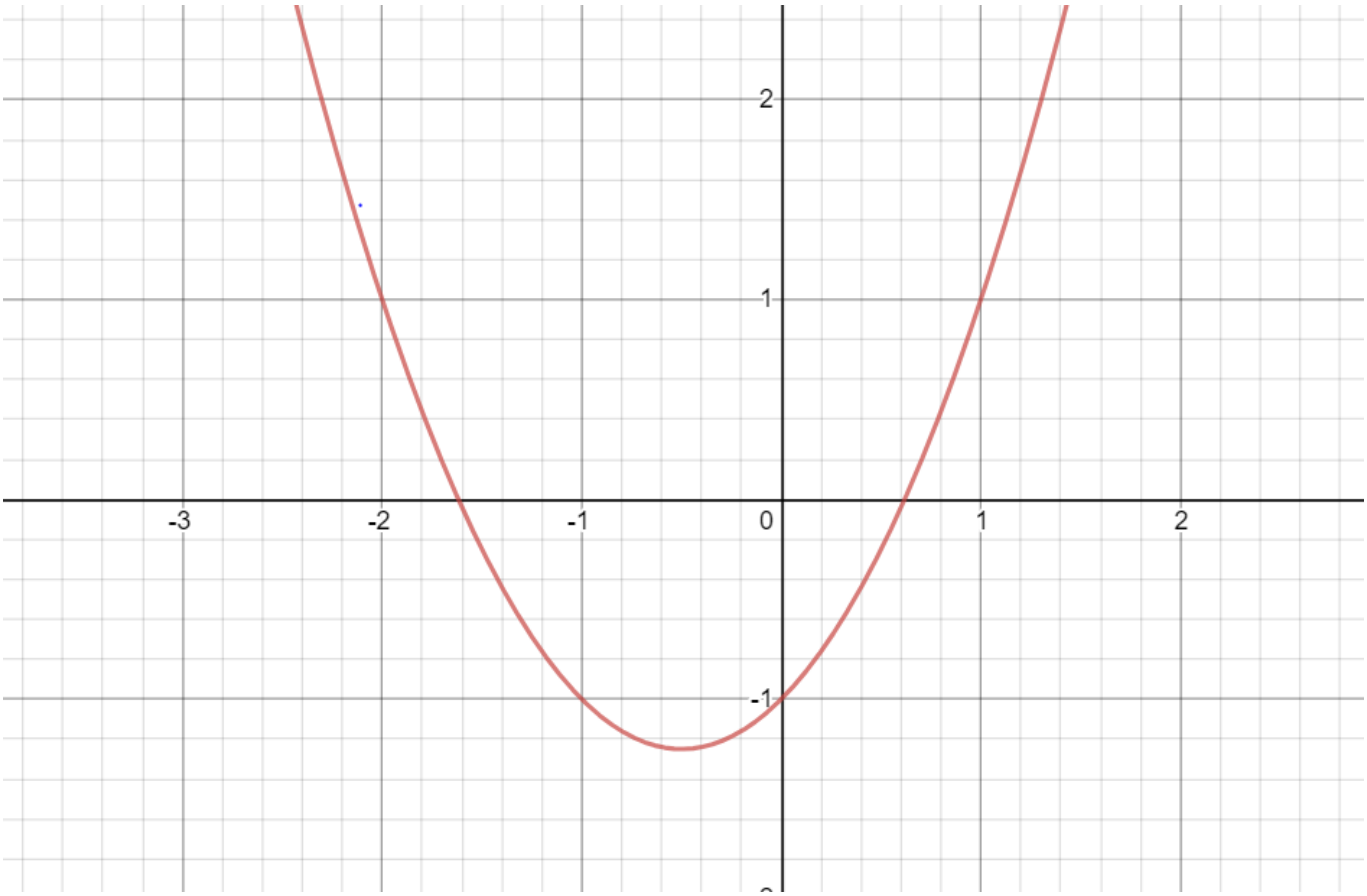
Jeżeli od pierwszego równania odejmiemy drugie to otrzymamy:

$$x_2^2 + x_2 - 1 = 0$$

Wtedy mogę podstawić $x_2 = x$ oraz $f(x) = x^2 + x - 1$ i układ równań przyjmuje postać:

$$f(x) = x^2 + x - 1 = 0$$

Dla takiego równania nieliniowego można spokojnie zastosować metodę Newtona-Raphsona. Musimy najpierw zobaczyć wykres $f(x)$ by móc stwierdzić, gdzie szukać pierwiastków równania. Wykres $f(x)$ widoczny jest poniżej:



Z wykresu widać, że powinniśmy spodziewać się dwóch rozwiązań, jednego w przedziale $[-2,-1]$ a drugiego w przedziale $[0,1]$. Należy sprawdzić czy owe przedziały spełniają wszystkie warunki metody Newtona-Raphsona. Pierwszy warunek jest oczywiście spełniony, ponieważ w owych przedziałach jest tylko jeden pierwiastek. Drugi warunek jest również spełniony, ponieważ z wykresu widać, że wartości na krańcach przedziału są przeciwnego znaku. Policzmy pierwszą i drugą pochodną:

$$f'(x) = 2x + 1$$
$$f''(x) = 2$$

Nietrudno zauważyć, że w każdym z przedziałów warunek stałego znaku pochodnych jest spełniony. Dla przedziału $[-2,-1]$ te znaki to odpowiednio ujemny i dodatni zaś dla przedziału $[0,1]$ dodatni i dodatni. Możemy zatem zastosować metodę Newtona-Raphsona dla każdego z przedziałów.

```
def f(x):
    return x**2+x-1

def f_der(x):
    return 2*x+1

newton_raphson(f,f_der,-2,-1,10)
```

```
Value of x after 1 iterations: x = -1.6666666666666667
Value of x after 2 iterations: x = -1.619047619047619
Value of x after 3 iterations: x = -1.618034447821682
Value of x after 4 iterations: x = -1.618033988749989
Value of x after 5 iterations: x = -1.618033988749895
Value of x after 6 iterations: x = -1.618033988749895
Value of x after 7 iterations: x = -1.618033988749895
Value of x after 8 iterations: x = -1.618033988749895
Value of x after 9 iterations: x = -1.618033988749895
Value of x after 10 iterations: x = -1.618033988749895
```

```
def f(x):
    return x**2+x-1

def f_der(x):
    return 2*x+1

newton_raphson(f,f_der,0,1,10)
```

```
Value of x after 1 iterations: x = 1.0
Value of x after 2 iterations: x = 0.6666666666666667
Value of x after 3 iterations: x = 0.6190476190476191
Value of x after 4 iterations: x = 0.6180344478216819
Value of x after 5 iterations: x = 0.6180339887499892
Value of x after 6 iterations: x = 0.6180339887498948
Value of x after 7 iterations: x = 0.6180339887498948
Value of x after 8 iterations: x = 0.6180339887498948
Value of x after 9 iterations: x = 0.6180339887498948
Value of x after 10 iterations: x = 0.6180339887498948
```

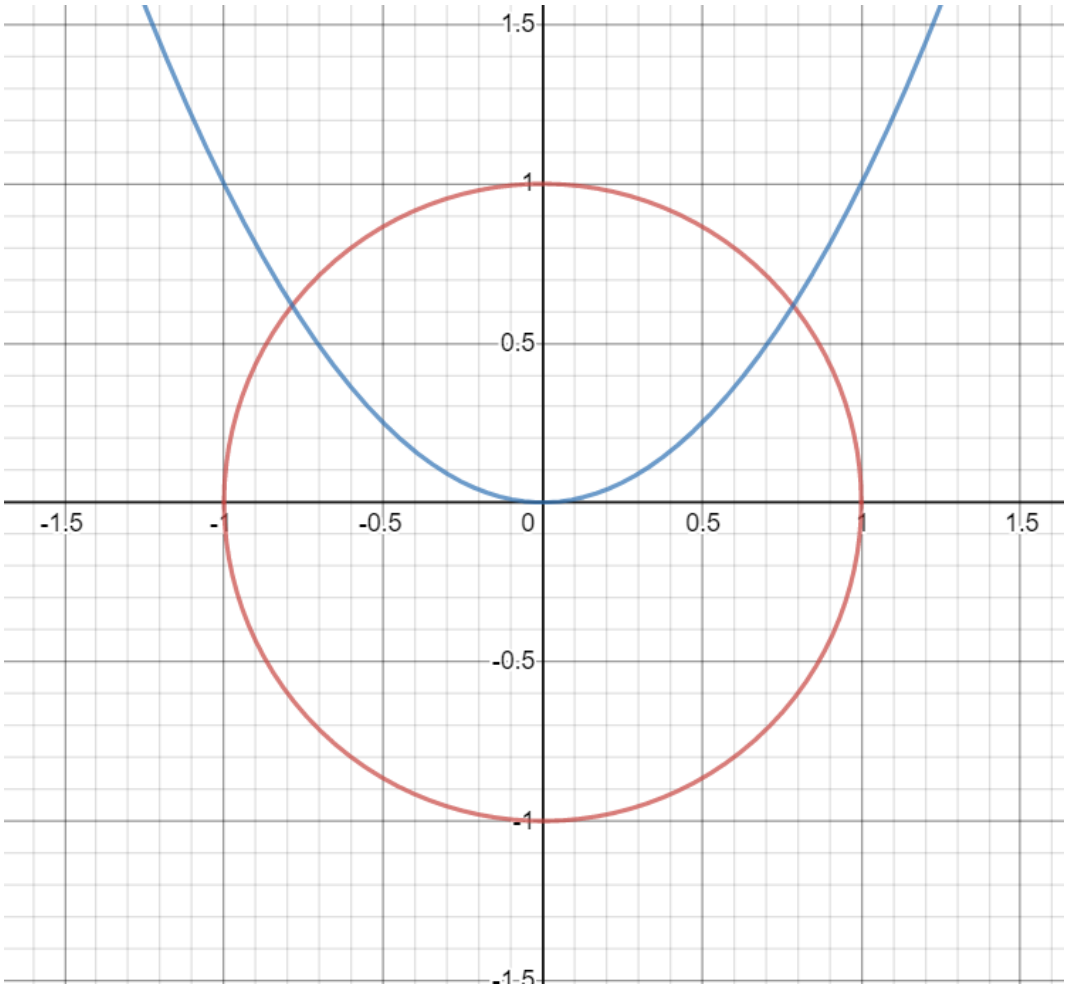
Znalezione wartości $x = x_2$ wynoszą $x_2 = -1.618033988749$ oraz $x_2 = 0.618033988749$. Odpowiadające im wartości x_1 można wyznaczyć z równania

$$x_1 = \pm\sqrt{x_2}$$

Zanim obliczymy wartości x_1 warto zauważyć, że z równania $x_1^2 - x_2 = 0$ wynika, że x_2 musi być nieujemne. Zatem odrzucamy wartość $x_2 = -1.618033988749$ i rozwiązaniem naszego układu równań są pary wartości:

$$x_1 = 0.786151377756,\ x_2 = 0.618033988749$$

$$x_1 = -0.786151377756,\ x_2 = 0.618033988749$$



Warto dodać, że owe rozwiązania są dokładne do 10 miejsc po przecinku co jest wartością zdecydowanie zadowalającą.

3.1 Wniosek

Wykonane ćwiczenie pokazuje, że metoda Newtona-Raphsona po prostych przekształceniach może skutecznie rozwiązywać układy równan nieliniowych.